Configure MySQL environment variables

Setting up the environment is necessary because our computers come with limited default settings. When new software is installed, it often cannot be recognized or operated with higher permissions. For some software like Python, Java, and certain Linux applications, environment configuration is needed to perform advanced operations.

MySQL Environment Configuration:

Open the command prompt (cmd) and enter **mysql -u root -p**.

Part Two:

This code is for the ESP8266 in the Arduino IDE. Replace the ssid and password with your own. To summarize, the setup section contains the configuration, the loop section contains the code to read the DHT sensor, and the HTTP code handles the normal data transmission process.

```
#include <ESP8266WiFi.h>  // Include the ESP8266 WiFi library for WiFi connection
#include <ESP8266HTTPClient.h>  // Include the ESP8266 HTTP client library for
HTTP requests
#include "DHT.h"  // Include the DHT sensor library for reading temperature and
humidity data

#define DHTPIN 2  // Define the data pin for the DHT sensor as pin 2
#define DHTTYPE DHT22  // Define the DHT sensor type as DHT22

const char* ssid = "whyhd";  // Define the WiFi network SSID
const char* password = "eoeoeoeo";  // Define the WiFi network password
const String serverName = "http://10.82.1.156:5000";  // Define the server
address (replace with your server IP address)

DHT dht(DHTPIN, DHTTYPE);  // Create a DHT sensor object, specifying the pin and
sensor type

void setup() {
  Serial.begin(9600);  // Initialize serial communication with a baud rate of
9600
  WiFi.begin(ssid, password);  // Start connecting to the WiFi network

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);  // Delay for 1 second
    Serial.println("Connecting to WiFi...");  // Print the WiFi connection status
```

```
  }

  Serial.println("Connected to WiFi");  // Print the successful connection status
  dht.begin();  // Initialize the DHT sensor
}

void loop() {
  if (WiFi.status() == WL_CONNECTED) {  // Check if WiFi is connected
    WiFiClient client;  // Create a WiFi client object
    HTTPClient http;  // Create an HTTP client object
    float h = dht.readHumidity();  // Read humidity data
    float t = dht.readTemperature();  // Read temperature data

    if (isnan(h) || isnan(t)) {  // Check if the data read is valid
      Serial.println("Failed to read from DHT sensor!");  // Print the failed
reading status
      return;  // Exit the function
    }

    // Create the request URL, including temperature and humidity data
    String url = serverName + "/update?sensor=dht22&temperature=" + String(t) +
"&humidity=" + String(h);

    http.begin(client, url); // Initialize the HTTP request with the new begin
method
    int httpCode = http.GET();  // Send the HTTP GET request

    if (httpCode > 0) {  // Check if the HTTP response code is greater than 0
      String payload = http.getString();  // Get the response content
      Serial.println("HTTP Response code: " + String(httpCode));  // Print the
HTTP response code
      Serial.println("Response: " + payload);  // Print the response content
    } else {
      Serial.println("Error on HTTP request, HTTP Response code: " +
String(httpCode));  // Print the HTTP request error
    }

    http.end();  // End the HTTP request
  } else {
    Serial.println("WiFi not connected");  // Print the WiFi not connected status
  }

  delay(2000);  // Delay for 2 seconds (adjust as needed)
}
```

Logical summary

- **In the setup function,** the code is responsible **for initializing serial port communication, connecting to the WiFi network, and initializing the DHT sensor.**
- **In the loop function,** the code is responsible **for continuously reading the data from the DHT sensor and sending the data to the server via HTTP GET requests**. **Reliable data transmission is ensured by checking the WiFi connection status and sensor data validity,** and a 2-second delay is added after each loop to control the request.

HTTP GET request is a method of requesting data from a server. For example, when you enter a URL in a browser and visit a web page, **the browser sends an HTTP GET request to the server, and the server returns the corresponding web page content.**

```
    // Create the request URL, including temperature and humidity data
    String url = serverName + "/update?sensor=dht22&temperature=" + String(t) +
"&humidity=" + String(h);
```

Concatenate the server address and sensor data into a complete URL. This URL tells the server what data we want to send.

```
http.begin(client, url); // Initialize the HTTP request with the new begin method
    int httpCode = http.GET();  // Send the HTTP GET request

    if (httpCode > 0) {  // Check if the HTTP response code is greater than 0
      String payload = http.getString();  // Get the response content
      Serial.println("HTTP Response code: " + String(httpCode));  // Print the
HTTP response code
      Serial.println("Response: " + payload);  // Print the response content
    } else {
      Serial.println("Error on HTTP request, HTTP Response code: " +
String(httpCode));  // Print the HTTP request error
    }

    http.end();  // End the HTTP request
```

The response code tells us whether the request is successful. If the response code is greater than 0, it means the request is successful, and we can get the data returned by the server and print it out. If the response code is not greater than 0, it means the request failed, and an error message is printed. Finally, close the HTTP connection and release resources.

The Python code is written in this way: put the URL, which corresponds to the Arduino URL. If the http code is 200, which is a reasonable situation, it will print the desired result; otherwise, it will print fail to retrieve data.

```
from flask import Flask, request  # Import Flask and request modules to create a server and handle requests
import pandas as pd  # Import pandas module for data processing and saving to Excel
import time  # Import time module for delay operations in the thread
from threading import Thread  # Import Thread module to create an independent thread
from datetime import datetime, timedelta  # Import datetime module to get the current time

app = Flask(__name__)  # Create a Flask application instance

# Initialize a dictionary to store timestamps, humidity, and temperature data
data = {'Timestamp': [], 'Humidity': [], 'Temperature': []}

def save_data_to_excel():
    while True:
        time.sleep(20)  # Save data every 20 seconds
        if data['Timestamp']:  # If there is data
            df = pd.DataFrame(data)  # Convert the dictionary to a pandas DataFrame
            df.to_excel('sensor_data.xlsx', index=False)  # Save the DataFrame to an Excel file without saving the index
            print('Data saved to Excel')  # Print a message indicating the data was saved successfully

@app.route('/update', methods=['GET'])  # Define a route and handler method to process GET requests
def update():
    sensor = request.args.get('sensor')  # Get the sensor type from the request
    temperature = request.args.get('temperature')  # Get the temperature data from the request
    humidity = request.args.get('humidity')  # Get the humidity data from the request

    # Add the timestamp, humidity, and temperature data to the dictionary
    data['Timestamp'].append(datetime.now().strftime("%Y-%m-%d %H:%M:%S"))  # Add the current timestamp
    data['Humidity'].append(humidity)  # Add the humidity data
    data['Temperature'].append(temperature)  # Add the temperature data
```

```
    print(f"Received data: Sensor={sensor}, Temperature={temperature},
Humidity={humidity}")  # Print the received data
    return "Data received"  # Return a response message

if __name__ == '__main__':  # Start a thread to periodically save data to the
Excel file
    save_thread = Thread(target=save_data_to_excel)  # Create a thread with the
target function save_data_to_excel
    save_thread.start()  # Start the thread
    app.run(host='0.0.0.0', port=5000)  # Start the Flask server, listening on
all IP addresses at port 5000
```

Code logic summary

- Created a Flask server to receive sensor data.
- Initialized a data dictionary to store timestamp, temperature, and humidity data.
- Defined a function save_data_to_excel, which saves the data in the data dictionary to an Excel file every 20 seconds.
- **Used Flask's routing function to define an /update route to receive GET requests and store sensor data in the data dictionary.**
- Used the thread save_thread to save data periodically to ensure that the main thread is not blocked waiting for data to be saved.
- **Started the Flask server, listening to port 5000 of all IP addresses, and handling incoming requests.**

```
@app.route('/update', methods=['GET'])  # Define a route and handler method to
process GET requests
```

This line of code tells Flask that when the server receives a GET request for a URL ending in /update, it should call the update function to handle the request.

```
def update():
    sensor = request.args.get('sensor')  # Get the sensor type from the request
    temperature = request.args.get('temperature')  # Get the temperature data
from the request
    humidity = request.args.get('humidity')  # Get the humidity data from the
request
```

request.args.get('sensor') gets the value of the parameter sensor from the URL's query string. Similarly, it gets the values of temperature and humidity. These values are sent from the client (such as ESP8266).

```
    # Add the timestamp, humidity, and temperature data to the dictionary
    data['Timestamp'].append(datetime.now().strftime("%Y-%m-%d %H:%M:%S"))  # Add
the current timestamp
    data['Humidity'].append(humidity)  # Add the humidity data
    data['Temperature'].append(temperature)  # Add the temperature data
```

datetime.now().strftime("%Y-%m-%d %H:%M:%S") gets the current time and formats it as a string. This timestamp is then added to the Timestamp list in the data dictionary.

```
    print(f"Received data: Sensor={sensor}, Temperature={temperature},
Humidity={humidity}")  # Print the received data
    return "Data received"  # Return a response message

if __name__ == '__main__':  # Start a thread to periodically save data to the
Excel file
```

This line of code checks whether the current script is being run directly. If the script is being run directly, then the value of __name__ is __main__.

```
    save_thread = Thread(target=save_data_to_excel)  # Create a thread with the
target function save_data_to_excel
    save_thread.start()  # Start the thread
```

Thread(target=save_data_to_excel) creates a new thread that will run the save_data_to_excel function. save_thread.start() starts the thread and lets it start running the save_data_to_excel function. This function saves data to an Excel file every 20 seconds.

```
    app.run(host='0.0.0.0', port=5000)  # Start the Flask server, listening on
all IP addresses at port 5000
```

app.run(host='0.0.0.0', port=5000) starts the Flask server and sets the server to listen on all IP addresses (0.0.0.0) on port 5000. The server will handle all HTTP requests sent to this address and port.

Hardware Connection

To connect the DHT22 temperature and humidity sensor to the ESP8266:

- **Connect the VCC pin of the DHT22 to the 3.3V power pin of the ESP8266:** Ensure that the DHT22 sensor is powered by the 3.3V provided by the ESP8266.
- **Connect the GND pin of the DHT22 to the GND pin of the ESP8266:** This will provide the necessary grounding.
- **Connect the data pin of the DHT22 to the D4 pin of the ESP8266:** D4 is a digital input/output pin on the ESP8266, corresponding to GPIO2. Connecting the data pin of the DHT22 to the D4 pin allows the ESP8266 to read temperature and humidity data from the DHT22 sensor.

With this connection setup, the ESP8266 can read the data transmitted by the DHT22 sensor through the D4 pin, enabling temperature and humidity monitoring.